УДК 004.41.42

Н. П. Цыганенко
Белорусский государственный технологический университет

СТАТИЧЕСКИЙ АНАЛИЗ КОДА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ КАК СРЕДСТВО ВЫЯВЛЕНИЯ ЕГО УЯЗВИМОСТЕЙ

В статье рассмотрен статический анализ исходного кода программ для мобильных платформ как одно из средств обеспечения безопасности мобильных систем. Проанализированы различные сценарии практического применения такого рода анализатора в качестве средства превентивной защиты. Проведено моделирование уязвимости типа SQL-Injection при помощи интегрированной среды разработки Microsoft Visual Studio 2013 Professional и языка программирования C#. Модель представлена в виде исходного кода. На основании полученной модели разработан алгоритм для выявления данного типа уязвимостей в исходных кодах программ мобильных приложений. Базируясь на информации о статическом анализе исходного кода программ, а также используя разработанный алгоритм выявления уязвимостей типа SQL-Injection, создан статический анализатор, выявляющий уязвимости такого рода. Анализатор написан на языке программирования C#. Представлена структурная схема главного модуля приложения. Для выполнения проверки условий, описанных алгоритмом выявления уязвимостей типа SQL-Injection, статический анализатор использует синтаксические деревья. При построении и обработке синтаксических деревьев задействован функционал одного из пространств имен для работы с C# библиотеки Roslyn – компилятор с открытым исходным кодом от компании Microsoft.

Ключевые слова: безопасность, мобильные системы, статический анализ, синтаксическое дерево, Roslyn.

N. P. Tsyganenko
Belarusian State Technological University

THE STATIC ANALYSIS OF MOBILE APPLICATIONS CODE AS VULNERABILITIES DETECTION METHOD

A static analysis of applications source code for mobile platforms as vulnerabilities detection method is considered. Various scenarios of practical usage of that kind of an analyzer are examined. The static analysis can be used as a preventive protection method. The SQL-Injection vulnerability was simulated using an integrated development environment Microsoft Visual Studio 2013 and the programming language C#. The model is presented in the form of a source code. Based on this simulation, we create an algorithm for the detecting of that type of vulnerabilities in source code of mobile applications. The static analyzer, which detects SQL-Injection vulnerabilities, is developed with usage of information about the static analysis of applications source code, as well as the derived algorithm for detecting that type of vulnerabilities. The analyzer is developed using the programming language C#. The block diagram of the main module of the application is illustrated. The static analyzer uses syntax trees to perform the test conditions described by detecting algorithm of SQL-Injection vulnerabilities. The syntax trees are constructed and processed with the use of functionality of one of the namespaces located in Roslyn libraries – an open source compiler from Microsoft.

Key words: security, mobile systems, static analysis, syntax tree, Roslyn.

Введение. Угрозы, которым подвержены мобильные платформы, схожи с угрозами, характерными для настольных систем, с некоторыми различиями, обусловленными особенностями мобильных устройств. Можно выделить следующие уровни угроз:

– физический; кража, несанкционированный доступ к информации; более легкое осуществление по сравнению с настольными компьютерами;

– аппаратный; закладки в аппаратной части телефона, установленные злоумышленником вредоносные или следящие устройства, использование информации от операторов связи;

– системный; закладки и уязвимости операционных систем; открытость операционных систем, с одной стороны, уменьшает вероятность закладок, с другой стороны, увеличивает фрагментацию рынка, что приводит к усложнению доставки обновлений системы и исправлений уязвимостей; закрытые операционные системы, в свою очередь, упрощают доставку обновлений и исправлений, но не дают гарантий отсутствия закладок;

– приложений; уязвимости и закладки в системных предустановленных приложениях и приложениях сторонних разработчиков;

– социальный; методы и средства социальной инженерии; самый трудный с точки зрения

обеспечения безопасности уровень, так как его методы основываются на психологии людей.

Наиболее перспективным и доступным для разработок и экспериментов является класс прикладных средств обеспечения безопасности. С одной стороны, это антивирусы и фаерволлы, которые защищают пользователя непосредственно во время работы с телефоном, с другой – анализаторы приложений, защищающие маркеты от загрузки злоумышленниками вредоносных приложений [1].

Большой частью проблемы безопасности мобильных систем является определение уязвимости приложений, которые пользователи устанавливают на свой смартфон. Основная доля пользователей мобильных устройств использует специализированные маркеты, которые распространяют приложения (App Store, Play Market, Яндекс Маркет, Win Store). Программное обеспечение в таких маркетах проходит проверку специалистами, но выявление уязвимостей и закладок – достаточно трудоемкий процесс. Возможно облегчить данный процесс, введя некоторую автоматизацию. Частью этой автоматизации могут быть специальные статические анализаторы, которые проверяют исходный код приложений на наличие уязвимостей [2].

Такими статическими анализаторами могут воспользоваться, в том числе сами разработчики приложений, если они используют сторонние библиотеки, в происхождении и надежности которых они не уверены.

Основная часть. Статический анализ кода – анализ исходных кодов приложения, производимый (в отличие от динамического анализа) без реального выполнения проверяемых программ. В подавляющем большинстве случаев статический анализ осуществляется над какой-либо версией исходного кода, обычно последней, хотя иногда анализу подвергается один из видов объектного кода, например P-код или MSIL-код. Термин обычно применяют к анализу, производимому специальным ПО [3].

Неавтоматизированной версией статического анализа можно назвать обзор кода. Обзор кода – один из самых старых и надежных методов выявления дефектов. Он заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению. В процессе чтения кода выявляются ошибки или участки кода, которые могут стать ошибочными в будущем. Также считается, что автор кода во время обзора не должен давать объяснений, как работает та или иная часть программы. Алгоритм работы должен быть понятен непосредственно из текста программы и комментариев. Если это условие не выполняется, то код должен быть доработан.

Чаще обзор кода работает лучше статического анализатора при условии, что реализуется опытными программистами. Единственный существенный недостаток методологии совместного обзора кода – это крайне высокая цена. Необходимо регулярно собирать нескольких программистов для обзора нового кода или повторного обзора кода после внесения рекомендаций. При этом программисты должны регулярно делать перерывы для отдыха. Если пытаться просматривать сразу большие фрагменты кода, то внимание быстро притупляется и польза от обзора кода быстро сходит на нет.

По этим причинам, с одной стороны, хочется, как можно чаще применять обзор кода. С другой – это слишком дорогостоящий и трудоемкий процесс. Компромиссным решением являются инструменты статического анализа кода. Они постоянно обрабатывают исходные тексты программ и выдают программисту рекомендации обратить повышенное внимание на определенные участки кода. Конечно, программа не заменит полноценного обзора кода, выполняемого коллективом программистов. Однако соотношение польза/цена делает использование статического анализа весьма полезной практикой, применяемой многими компаниями.

Одним из главных преимуществ статического анализа является возможность существенного снижения стоимости устранения дефектов в программе. Чем раньше ошибка выявлена, тем меньше стоимость ее исправления. Так, согласно опытным данным, исправление ошибки на этапе тестирования обойдется в 10 раз дороже, чем на этапе конструирования программы (написания кода) [4].

В зависимости от используемого инструмента глубина анализа может варьироваться от определения поведения отдельных операторов до анализа, включающего весь имеющийся исходный код. Способы использования полученной в ходе анализа информации также различны – от выявления мест, возможно содержащих ошибки, до формальных методов, позволяющих математически доказать какие-либо свойства программы (например, соответствие поведения спецификации). Некоторые считают программные метрики и обратное проектирование формами статического анализа. Получение метрик и статический анализ часто совмещаются, особенно при создании встраиваемых систем, но все-таки это разные процессы.

Статический анализ не гарантирует корректного выявления ошибок и уязвимостей. Возможны ложные (false positive) и ложноотрицательные срабатывания (false negative) анализатора. По этой же причине статический

анализатор в общем случае не предназначен для исправления найденных ошибок и уязвимостей. Он предупреждает программиста о подозрительных и потенциально проблемных участках кода [5].

Простейший пример статического анализа можно продемонстрировать при помощи фрагмента кода на языке C#:

```
int x;
int y = x + 2;
```

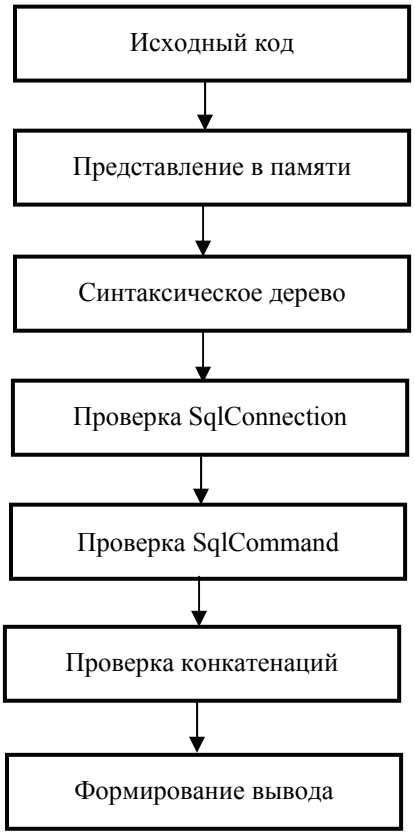
Статический анализатор для данного кода выведет сообщение о том, что переменная *x* не инициализирована, хотя код синтаксически правилен. Выявлением таких очевидных ошибок обычно занимается компилятор.

Для построения статического анализатора, который будет детектировать уязвимости в исходных кодах программ, нужен алгоритм выявления. Чтобы составить необходимый алгоритм, вначале требуется промоделировать уязвимость, которую анализатор будет впоследствии выявлять. В качестве такой уязвимости промоделируем SQL-Injection:

```
var connection = new SqlConnection(ConntectionString);

const string parameterFromUser
    = "5; drop table Users";
const string query
    = "select * from News where Id = "
    + parameterFromUser;

var command = new SqlCommand(query);
RunCommands(connection, command);
. . .

static void RunCommands(
    IDbConnection connection,
    params IDbCommand[] commands)
{
    connection.Open();
    foreach (var command in commands)
    {
            command.Connection
                = connection;
            command.ExecuteNonQuery();
    }
    connection.Close(); }
```

Данная модель показывает использование неэкранированного пользовательского ввода при выполнении запросов к базе данных, что может привести к нарушению целостности данных, захвату конфиденциальной информации или полному разрушению базы данных.

Структурная схема статического анализатора, разработанного по алгоритму выявления созданной уязвимости, представлена на рисунке.



Структурная схема статического анализатора

По результатам анализа построенной модели уязвимости типа SQL-Injection с использованием языка C# алгоритм выявления можно записать следующим образом:

1) применяется ли в коде одна из реализаций IDbConnection;

2) используется ли в коде одна из реализаций IDbCommand;

3) применяется ли в коде неэкранированный пользовательский ввод.

При построении статического анализатора будем считать, что если код отвечает положительно на все вопросы алгоритма выявления, то такой код содержит уязвимость типа SQL-Injection.

Для проверки условий алгоритма выявления статический анализатор будет использовать синтаксические деревья. Строить и обрабатывать их можно с помощью компилятора с открытым исходным кодом Roslyn от компании Microsoft. Он предоставляет специальное пространство имен CodeAnalysis, содержащее необходимые для работы классы и функции. Например, для проверки наличия экземпляра конкретного класса можно воспользоваться специальными классами Object Creation Expression

Syntax и Identifier Name Syntax. Первый позволяет найти в синтаксическом дереве все операции создания объектов, а второй – определить тип создаваемого объекта.

Разработанный статический анализатор работает следующим образом:

– считывает файл с исходным кодом приложения с диска в память;

– преобразует строковое представление файла в памяти в синтаксическое дерево с помощью средств компилятора Roslyn;

– проверяет полученное синтаксическое дерево на соответствие условиям, заданным алгоритмом выявления уязвимости типа SQL-Injection;

– по результатам проверки выдает решение о наличии или отсутствии уязвимости типа SQL-Injection.

Реализованный статический анализатор корректно срабатывает на имеющейся модели уязвимости типа SQL-Injection.

Заключение. Описана модель уязвимости типа SQL-Injection. На основании анализа этой модели разработан алгоритм выявления уязвимостей данного типа. Используя разработанный алгоритм выявления, создан статический анализатор на основе синтаксических деревьев, специализирующийся на уязвимостях типа SQL-Injection. Дальнейшее развитие алгоритма выявления может быть направлено в сторону уменьшения вероятности ложного срабатывания. Созданный статический анализатор может использоваться при проверке исходных кодов приложений до загрузки их сторонними разработчиками в специализированные маркеты.

Литература

1. Furnell S. Mobile Security. A Pocket Guide. Ely: IT Governance, 2009. P. 49–65.

2. Drake J. J. Android Hacker’s Handbook. Indianapolis: Wiley, 2014. P. 104.

3. Bergman N. Hacking Exposed: Mobile Security Secrets & Solutions. NY: Mc Graw Hill, 2013. P. 30–35.

4. Chess B. Secure Programming with Static Analysis. Boston: Addison-Wesley Professional, 2007. P. 501–503.

5. Bounlanger J. Static Analysis of Software: The Abstract Interpretation. Indianapolis: Wiley, 2011. P. 258–260.

References

1. Furnell S. Mobile Security. A Pocket Guide. Ely, IT Governance, 2009, pp. 49–65.

2. Drake J. J. Android Hacker’s Handbook. Indianapolis, Wiley, 2014, pp. 104.

3. Bergman N. Hacking Exposed: Mobile Security Secrets & Solutions. NY, Mc Graw Hill, 2013, pp. 30–35.

4. Chess B. Secure Programming with Static Analysis. Boston, Addison-Wesley Professional, 2007, pp. 501–503.

5. Bounlanger J. Static Analysis of Software: The Abstract Interpretation. Indianapolis, Wiley, 2011, pp. 258–260.

Информация об авторе

Цыганенко Никита Павлович – магистр технических наук. Белорусский государственный технологический университет (220006, г. Минск, ул. Свердлова, 13а, Республика Беларусь). E-mail: nikitatsyganenko@belstu.by

Information about the author

Tsyganenko Nikita Pavlovich – master of engineering. Belarusian State Technological University (13a, Sverdlova str., 220006, Minsk, Republic of Belarus). E-mail: nikitatsyganenko@belstu.by

Поступила 26.02.2015